Evolúciós algoritmusok 3. előadás

Genetikus operátorok permutáció reprezentációra

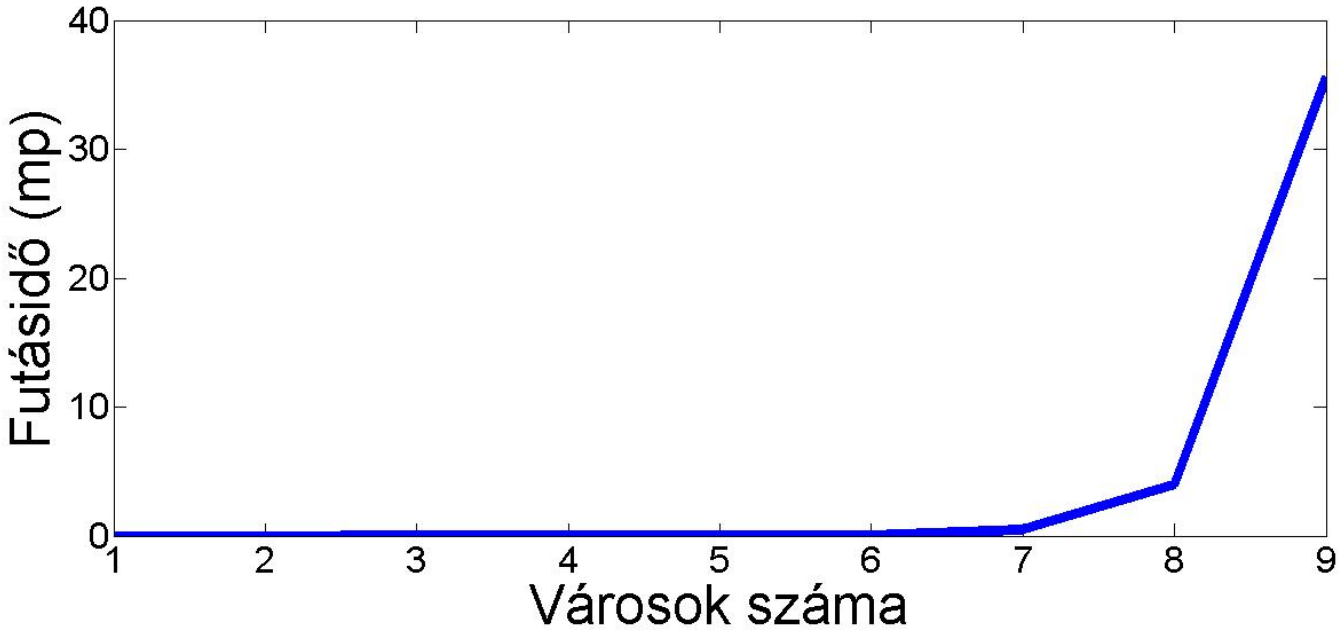
Utazó ügynök probléma

Adott n város, és ismert, hogy bármely két város között mennyi idő az utazás. Kérdés, hogyan lehet az összes várost a lehető legrövidebb idő alatt meglátogatni.

A probléma úgy is megfogalmazható, hogy adott egy G teljes élsúlyozott gráf, és keresem a minimális összsúlyú Hamilton-kört.

A naiv algoritmus végigmegy a csúcsok összes lehetséges sorrendjén, mindegyik sorrendre kiszámítja az élsúlyok összegét. Ez $(n-1)!$ féle sorrend, mindegyik sorrenden n összeadás.

Futásidő brute force megvalósításra



Utazó ügynök probléma dinamikus programozással

Legyenek a G gráf csúcsai az 1, 2, . . . , n számokkal címkézve, és jelöljük az i-edik és j-edik csúcs közötti élsúlyt a_{ij}-vel.

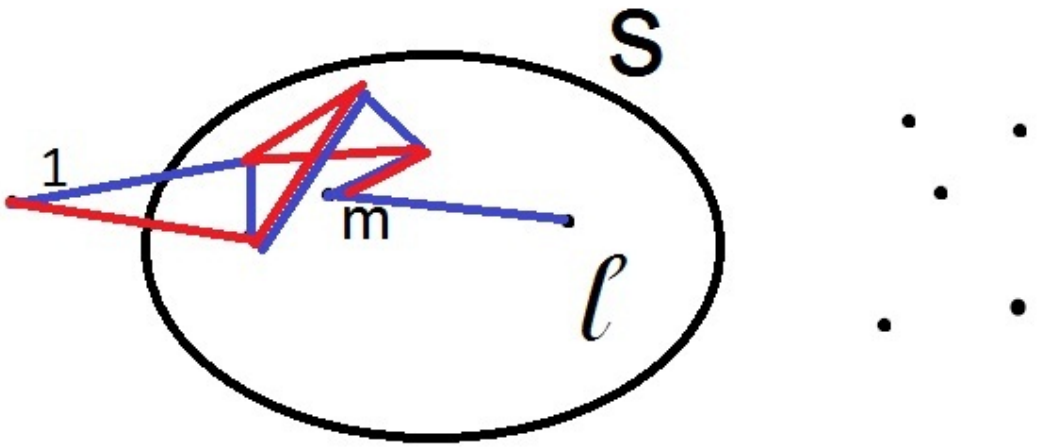
Definíció

Legyen $S \subset \{2, 3, \ldots, n\}$, $I \in S$, ekkor $C(S, \ell)$-vel jelöljük a legrövidebb olyan út hosszát, mely az 1-ből indul, minden S-beli elemet meglátogat, és az utolsó meglátogatott város ℓ.

Ezen $C(S, \ell)$ mennyiségek segítségével egy rekurzív algoritmust adhatunk a legrövidebb Hamilton-kör hosszának meghatározására (Karp-Held algoritmus).

A rekurzió

- $C(\{\ell\}, \ell) = a_{1\ell}$, $\forall \ell \in S$
- Ha $|S| > 1$ $C(S, \ell) = \min_{m \in S \setminus \{\ell\}} (C(S \setminus \{\ell\}, m) + a_{m\ell})$
- A minimális kör hossza $\min_{\ell \in \{2,3,\ldots n\}} (C(\{2, 3, \ldots, n\}, \ell) + a_{\ell 1})$



Utazó ügynök probléma dinamikus programozással

A rekurzió lépésszámát úgy határozhatjuk meg, hogy S mérete szerint esetekre bontunk. Legyen az adott szinten $|S| = k$. Ekkor $\binom{n-1}{k}$-féle S létezik összesen, ℓ-et k-féle módon, ezután m-et $k-1$-féle módon választhatom ki.

$$\left(\sum_{k=1}^{n-1} k(k-1)\binom{n-1}{k}\right) + n - 1 =$$

$$\left(\sum_{k=2}^{n-3} \binom{n-3}{k-2}(n-1)(n-2)\right) + n + (n-1) =$$

$$(n-1)(n-2)2^{n-3} + (2n-1) = O(n^2 2^n)$$

A kétféle algoritmus összehasonlítása

Láttuk, hogy a rekurzív algoritmus is (szuper)exponenciális idejű.
És mennyivel jobb mint a brute-force?

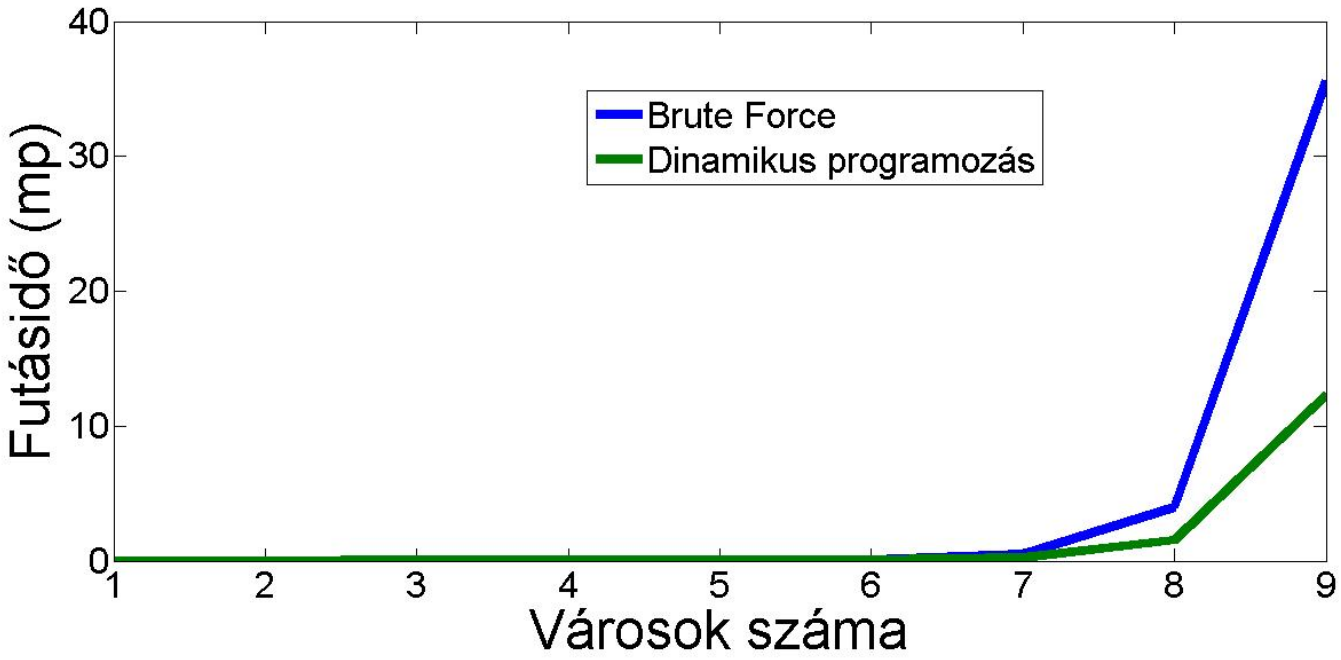
Tétel

Striling-formula

$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

Ez alapján a brute-force nagyságrendileg rosszabb.

Futásidő utazó ügynök problémára dinamikus programozással



Eldöntési nyelvek

Egy véges ábécé feletti véges szavak (egy tetszőleges) halmazát nyelvnek nevezzük. Azt a feladatot, hogy döntsük el, hogy egy szó benne van-e az adott nyelvben nevezzük eldöntési feladatnak.

Példa: Legyenek $x_1, x_2, \ldots$ logikai változók (értékük 0 vagy 1). Ezen változókból felépülő véges Boole-formulák közül azok vannak benne a SAT-nyelvben, amelyek igazzá tehetők valamely x_i választásokkal. Így például a $x_1 \wedge \neg x_1$ nincs benne, de a $(x_1 \wedge x_2) \vee \neg x_1$ igen.

Problémaosztályok

Definíció

Egy nyelv P-beli, ha polinom időben eldönthető, hogy egy szó benne van-e.

Ezen feltételtelen gyengítve kapjuk az NP osztályt. Itt formálisan nem definiáljuk NP-t, helyette az osztályt jellemző tételt mondjuk ki.

Tétel

(Tanú-tétel) Egy eldöntési nyelv NP-beli, ha az arra vonatkozó tanú, hogy egy szó benne van-e, polinom időben ellenőrizhető.

Példa

Veszem az összes irányítatlan gráfot. Egy gráf benne van a H nyelvben, ha van benne Hamilton-kör. Ha valaki tud mutatni egy Hamilton-kört, az tanúsítja, hogy a gráf H-beli. Egy Hamilton-kört pedig polinom időben tudunk ellenőrizni, hiszen ha a gráf n csúcsú, akkor a szomszédsági mátrixban n elem nemnulla voltát ellenőrizzük.

Definíció

Egy nyelv NP-nehéz, ha minden más NP-beli probléma visszavezethető rá. NP-teljes, ha NP-nehéz és maga is NP-beli.

Híres sejtés, hogy $P \neq NP$.

Néhány nevezetes NP-teljes probléma

- A 3 színnel színezhető gráfok nyelve
- Egy gráfban annak eldöntése, hogy van-e k csúcsú teljes részgráfja
- Egy gráfban van-e Hamilton-kör
- Egy súlyozott gráfban van-e k-nál nem hosszabb Hamilton-kör
- A hátizsákpakolás azon speciális esete, ahol $s_i = v_i$ és az a kérdés, hogy van-e pontosan k értékű pakolás
- Adottak $d_1, d_2, \ldots, d_k$ feladatok és T szám, mindegyiket egységnyi időbe kerül elvégezni. Van rajtuk egy részben-rendezés: $d_1 \prec d_2$ azt jelenti, hogy d_1-t kell előbb elvégezni, mint d_2-t. Mindegyik feladathoz tartozik továbbá egy $h(d_i)$ határidő. A kérdés, hogy van-e a munkáknak olyan sorrendje, amelyben legfeljebb T munka nem készül el határidőre?

A permutáció reprezentáció

Mivel NP-teljes probléma, ezért szeretnénk az Utazó Ügynök feladatot genetikus algoritmussal megoldani. A természetes reprezentáció egy n csúcsú gráfra az $1, 2, \ldots n$ számok egy sorrendje (permutációja). Ez a többi ütemezési feladathoz is alkalmas lesz.

A problémát a genetikus operátorok megtervezése jelenti. Hogyan tudjuk képezni egy permutáció mutációját? És két permutáció keresztezését?

Jelölések

Adott ez $1, 2, \ldots n$ számok egy sorrendje (permutációja).
Ezt a továbbiakban kerek zárójelbe tesszük, például: (3214) és Π-vel jelöljük.

Két lehetséges kódolás elterjedt a permutációk leírására: az elsőben az i. helyen a $\Pi(i)$ szerepel. A másodikban az lista i. eleme azt jelenti, hogy az i. elem hányadik a sorban.

Példa: Vegyük az A,B,C,D elemeket. Ekkor az első jelölés szerint a [3,1,2,4] lista az CABD sorrendet jelenti, a második szerint a BCAD-t. Az előadás további részében az első jelölést használjuk.

Tipikus feladatok

Két különböző típusú feladatnál jön szóba a potenciális megoldások kódolására a permutáció reprezentáció.

Az első típus az ütemezési feladatok, például amikor azt kell megtervezni, hogy milyen sorrendben gyártsunk le tárgyakat. Ha az A tárgyat a B és C előtt kell legyártani, és ezen hármat a D előtt, akkor az [1 2 3 4] és [1 3 2 4] fitnesze közel lesz egymáshoz, míg a [4 1 2 3]-től tőlük meglehetősen távol.

A második típus a „szomszédsági", mint például az utazó ügynök feladat. Például a [1 2 3 4] fitnesze meg fog egyezni a [4 1 2 3] fitneszével ezen feladatnál. Itt tehát az elemek kapcsolata a lényeges, nem a sorban elfoglalt pozíciója.

Mutáció operátorok

Permutáció esetén nem lehet csupán egyetlen gént megváltoztatni, ezért új mutáció operátorra van szükség. Ilyenkor az egyszerű genetikus algoritmusokkal ellentétben nem annak a valószínűségéről beszélhetünk, hogy egy gén mutálódik, hanem arról, hogy egy egyed.

Először az ütemezési feladatoknál használatos három operátort nézzük át. Ezek kis változást eredményeznek a sorrendben.

Mutáció operátorok

Csere mutáció (Swap mutation): Két elemet cserélünk ki.
Példa: (5 3 1 9 2 5 7 4 8) $\rightarrow$ (5 3 4 9 2 5 7 1 8).

Beszúrás mutáció (Insert mutation)
Kiválasztunk két véletlen gént, majd a másodikat az első mellé eltoljuk. Példa: (5 3 1 9 2 5 7 4 8) $\rightarrow$ (5 3 1 4 9 2 5 7 8) lesz.

Keverés mutáció (Scramble mutation)
Két véletlen pozíció közötti részt teljesen összekeverek, azaz veszünk egy megfelelő hosszúságú (rövid) véletlen permutációt, majd ezt alkalmazzuk a kiválasztott szakaszra.
Példa: (5 3 1 9 2 5 7 4 8) egy 4 hosszú szakasz. A keveréshez szükség van egy 4 elemű permutációra, vegyük például (3241)-et. Ekkor az eredmény: (5 3 2 9 5 1 7 4 8).

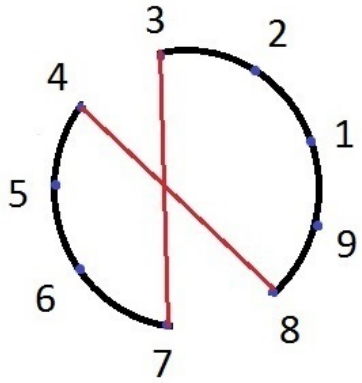
Mutáció operátorok

A következő mutáció operátor ajánlott szomszédsági feladatokhoz, mert ez okozza a lehető legkisebb változtatást.

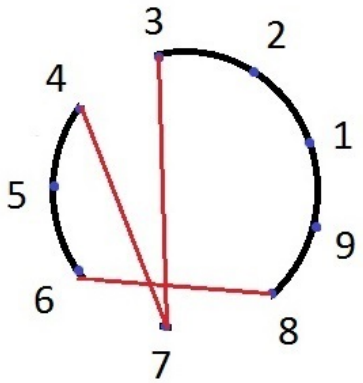
Inverzió mutáció (Inversion mutation) Kiválasztunk két véletlen pozíciót, majd a köztük lévő elemek sorrendjét megfordítjuk.
Példa: (6 3 1 9 2 5 7 4 8) $\rightarrow$ (6 3 5 2 9 1 7 4 8)

Mutáció operátorok

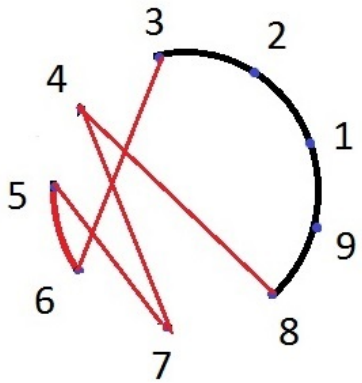
(123456789) INV (123765489)



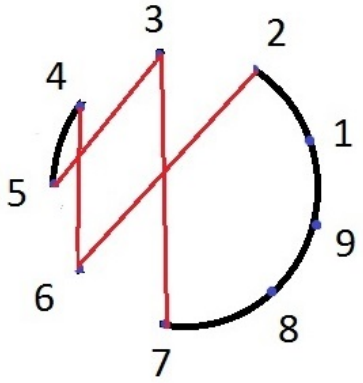
(123456789) INS (123745689)



(123456789) SCR (123657489)



(123456789) SWAP (126453789)



Keresztezés operátorok

A keresztezés operátorok célja, hogy a szülőkre hasonlító egyedeket hozzanak létre. Azt, hogy milyen tulajdonságot szeretnénk megőrizni, a feladat jellege diktálja.

Valamennyi operátorra jellemző, hogy két szülőből (többnyire a sorrendjük is számít) egy utódot hoz létre.

Két-két operátorral ismerkedünk meg, először a sorrend-típusú feladatokra kifejlesztetteket.

Order crossover (OX)

Ezen keresztezés az elemek egymáshoz viszonyított sorrendjét őrzi meg.

- Kiválasztunk két véletlen pozíciót, a közé eső szakasz lesz a megfeleltetési szakasz.
- Az utódba bemásoljuk p1 megfeleltetési szakaszát
- p2 még fel nem használt alléljait a második törési pozíciótól kezdve folytonosan átmásoljuk az utódba

Példa

Vegyük szülőknek a p1=(123456789) és p2=(937826514) permutációkat. Legyen a megfeleltetési szakasz a 4.-7. pozícióig.

Ekkor p1 közepét vesszük ki (123456789)
p2-ből kihúzzuk ezen elemeket: (937826514)

Bemásoljuk az utódba p1 megfeleltetési szakaszát: (␣␣␣4567␣␣)
majd folytonosan p2 megmaradt elemeit a 8. pozíciótól kezdve:
(382456719)

Cycle crossover (CX)

Ezen operátor megőrzi a szülőkben szereplő elemek pozíciójának abszolút helyét. A két szülőt körökre bontjuk úgy, hogy egy körben ugyanazon allélok szerepelnek mindkét szülőben, csak eltérő sorrendben. Jelöljük a két szülőt p1-el illetve p2-vel, ekkor a köröket a következő algoritmussal keressük meg:

- Kiindulunk a p1 első még fel nem használt alléljából, ez lesz a kezdőallél.
- Megkeressük p2 ugyanezen pozíciójában lévő alléljét, jelöljük i-vel
- Megkeressük i elem pozícióját p1-ben
- Addig ismételjük az előző két lépést, míg vissza nem érünk a kezdőallélhoz p1-ben; ekkor lesz kész egy kör

Példa

Vegyük szülőknek a p1=(123456789) és p2=(937826514) permutációkat. Ekkor az első kör:

(123456789)	(123456789)	(123456789)	(123456789)
(937826514)	(937826514)	(937826514)	(937826514)

A második kör:

(123456789)	(123456789)	(123456789)	(123456789)
(937826514)	(937826514)	(937826514)	(937826514)

A harmadik kör az egyedül maradt 6-os pozícióból fog állni. Ezután az utódot a p1 első, harmadik, ötödik, ... körében, és a p2 második, negyedik, ... körében lévő elemeiből kapjuk: (137426589).

Partially mapped crossover (PMX)

Ezen operátort a szomszédsági típusú feladatokra fejlesztették ki, célja, hogy minél több kapcsolatot megőrizzen a szülő permutációkból. A következőképpen működik:

- Vegyünk két véletlen pozíciót, ezek határozzák meg a megfeleltetési szakaszokat a szülőkben
- Másoljuk át p1-ből a megfeleltetési szakaszt (pozíciót megőrizve) az utódba
- Egyesével haladjunk végig p2 megfeleltetési szakaszán, és keressük ezen elemek helyét az utódban
- Az üresen maradt helyekre másoljuk be p2 elemeit (pozíciót megtartva)

Partially mapped crossover (PMX)

p2 megfeleltetési szakaszbeli elemeinek elhelyezése az utódban:

- Ha az adott elem már szerepel p1 megfeleltetési szakaszában, akkor nincs teendőnk
- Ha nem szerepel, akkor helyet kell keresnünk neki az utódban. Tegyük fel, hogy p2(j)-ről van szó
- Nézzük meg p1(j) értékét, majd keressük meg ezen érték helyét p2-ben, legyen ez k. Ha ez a pozíció nincs a megfeleltetési szakaszban, akkor erre a helyre másoljuk p2(j)-t. Ha benne van, akkor p1(k) veszi át p1(j) helyét addig, amíg nem találunk a megfeleltetési szakaszon kívülre eső helyet, ahová végül bemásoljuk p2(j)-t.

Példa

Vegyük a p1=(123456789) és p2=(937826514) permutációkat, és legyen a megfeleltetési szakasz a 4.-7. pozíciók.

Ekkor p1=(123456789) és p2=(937826514). Most az utód: (␣␣␣4567␣␣)

Vegyük sorban p2 megfeleltetési szakaszbeli elemeit. A 8-as nem szerepel p1 megfeleltetési szakaszában, ezért megnézzük p1(4)-et, ez most 4. A 4-es érték p2-ben a 9. pozícióban szerepel. Mivel ez kívül esik a megfeleltetési szakaszon, ezért az utód: (␣␣␣4567␣8)

Példa (folyt.)

A következő elem p2 megfeleltetési szakaszában a 2-es. Ennek is helyet kell keresnünk. p1(5)=5, az 5 a p2-ben a 7. pozícióban van, ami a megfeleltetési szakasz része, ezért ide nem tudjuk beilleszteni 2-t. Helyette megkeressük p1(7)=7-et p2-ben, az a 3. pozícióban van, ide másoljuk 2-t: (␣␣2 4567 ␣8).

A maradék kettő elem p2 megfeleltetési szakaszából 6 és 5, de mindketten benne vannak p1 megfeleltetési szakaszában is, így velük nincs teendő.

Most az üresen maradt helyekre beillesztjük p2 elemeit: (93245671 8).

A példából leolvasható, hogy az utód 6 éle szerepel valamelyik szülőben, viszont van egy olyan él is, a {7-8} amely mindkettő szülőben jelen van, mégsem szerepel az utódban.

Edge crossover (EX)

Ezen operátor megőrzi azon éleket, amelyek mindkét szülőben jelen vannak, továbbá minél több élet igyekszik a szülők egyikéből átvenni az utódba.

A szülőpárhoz először is létrehozunk egy közös éllistát, amelyben minden elemhez szerepelnek mindkét szülőbeli szomszédai, külön jelölve, ha valamelyik csúcs mindkét szülőben szomszéd.

Edge crossover (EX)

Az utódot a következő algoritmussal építjük fel:

- Elsőnek egy véletlen elemet választunk, ő lesz az aktuális csúcs
- Minden másik elem éllistájából kitöröljük az aktuális elemet
- Ha van az aktuális elem éllistájában közös csúcs, akkor ő lesz a következő aktuális elem (ha több ilyen van, akkor véletlenszerűen választunk)
- Ha nincs közös csúcs, akkor az lesz a követlező aktuális elem, akinek a legrövidebb a listája (ha több ilyen van, akkor véletlenszerűen választunk)
- Ha üres listához érkezek, akkor véletlenszerűen választunk a megmaradt csúcsok közül

Példa

Például vegyük a p1=(123456789) és p2=(937826514) permutációkat.

1	2,4,5,9	1	2,4,5,9				
2	1,3,6,8	2	3,6,8	2	3,6,8	2	3,8
3	2,4,7,9	3	2,4,7,9	3	2,4,7,9	3	2,4,7,9
4	1,3,5,9	4	3,5,9	4	3,9	4	3,9
5	1,4,6	5	4,6	5	4,6		
6	2,5,7	6	2,5,7	6	2,7	6	2,7
7	3,6,8	7	3,6,8	7	3,6,8	7	3,8
8	2,7,9	8	2,7,9	8	2,7,9	8	2,7,9
9	1,3,4,8	9	3,4,8	9	3,4,8	9	3,4,8
vv	(1)	ll	(15)	kcs	(156)	vv	(1562)

Példa (folyt.)

2	3,8						
3	4,7,9	3	4,7,9	3	4,9	3	4,9
4	3,9	4	3,9	4	3,9	4	9
7	3,8	7	3	7	3		
8	7,9	8	7,9				
9	3,4,8	9	3,4	9	3,4	9	4
II	(15628)	kcs	(156287)	II	(1562873)	II	(15628739)

Végül az utód az (156287394) permutáció lesz.